

INICIAR SESIÓN

NUESTROS PLANES

TODOS LOS  
CURSOS

FORMACIONES

CURSOS

PARA  
EMPRESAS

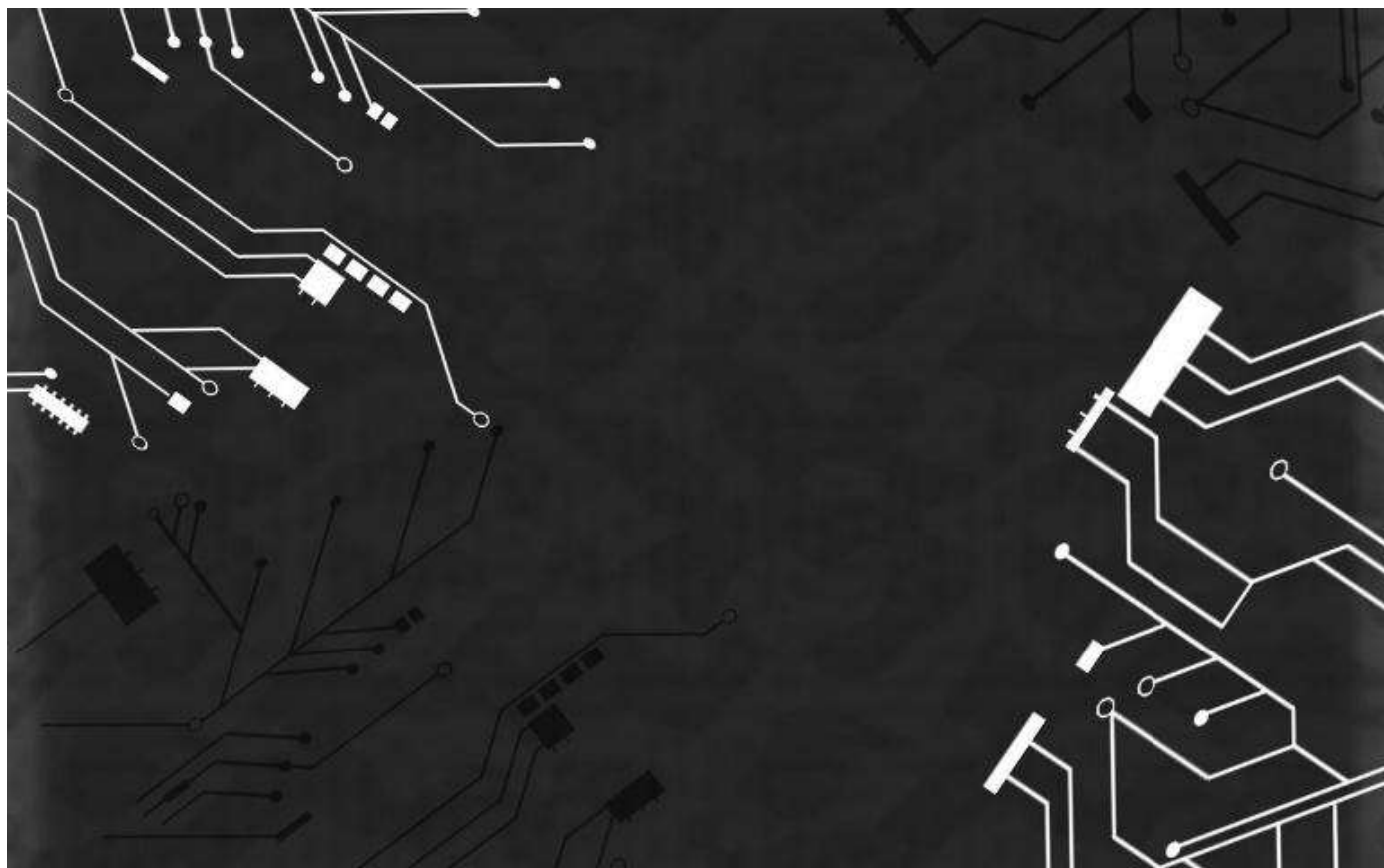
ARTÍCULOS DE TECNOLOGÍA &gt; DEVOPS

# Comenzando con Git: aprendiendo a versionar



Alex Felipe

29/07/2021



[Mira cómo usar git](#) y aplicar el control de versiones a sus proyectos. Estoy en un proyecto para un pequeño mercado y ahora estoy desarrollando una página web en HTML para listar todos los productos del mercado. Al principio obtuve el siguiente resultado:

## Todos os produtos

Refrigerante - R\$ 5,99
Sabão em pó - R\$ 9,99
Arroz - R\$ 10,99

Al mostrarle al cliente esta primera versión, dijo lo siguiente:

*"Es genial, sin embargo, sería mejor si al pulsar en un producto, apareciera una descripción y una cantidad".*

Según la necesidad del cliente, hice algunos ajustes en la página que enumera todos los productos, mira cómo se ve:


## Todos os produtos

Refrigerante - R\$ 5,99
<b>Descrição:</b> Dolly Guaranará o melhor! <b>Quantidade:</b> 100
Sabão em pó - R\$ 9,99
Arroz - R\$ 10,99

*"¡Está sirviendo muy bien para lo que necesito! Sin embargo, todavía es difícil saber qué es el producto con solo leerlo ... En otras palabras, quiero que también agregue una imagen del producto dentro de esta casilla de descripción y cantidad.*

Teniendo en cuenta esta nueva necesidad del cliente, hice los ajustes y obtuve este resultado:

# Todos os produtos

Refrigerante - R\$ 5,99	
	<p>Descrição: Dolly Guaraná o melhor!</p> <p>Quantidade: 100</p>
Sabão em pó - R\$ 9,99	
Arroz - R\$ 10,99	

Se lo mostré nuevamente al cliente, y él respondió lo siguiente:

*"Quedó genial la lista de productos con descripción, imagen y cantidad. Pero empecé a darme cuenta de que tiende a quedar con demasiada información visual, es decir, es mejor dejar la lista como estaba desde la **primera vez** que me mostraste, luego crea una nueva página que contiene la información del producto: imagen, descripción y cantidad ... "*


Ten en cuenta que desde el inicio del proyecto, **estamos sujetos a cambios constantes**. Por lo tanto, incluso si le entregamos la feature a nuestro cliente de la manera que solicita, todavía existe el riesgo de que quiera algo más. Además, ten en cuenta que también corremos el riesgo de que nuestro cliente simplemente nos diga que quería el proyecto como estaba antes, **algo que hicimos hace mucho tiempo** y que, dependiendo de la cantidad de cambios, es muy difícil revertir...

Pensando en todos estos detalles, ¿qué podemos hacer para evitar este tipo de situación?

## Aplicando control de versiones

Básicamente, necesitamos, de alguna manera, **guardar los estados de nuestro proyecto**, es decir, hacer con que cada cambio que entregamos sea una versión, por ejemplo, dividiendo en diferentes carpetas. Entonces tendríamos:

  
01-lista-simples

  
02-lista-com-  
descricao-e-  
quantidade

  
03-lista-com-  
imagem-descricao-  
e-quantidade

- **01-lista-sencilla:** primera entrega con una lista sencilla.
- **02-lista-con-descripción-y-cantidad:** segunda entrega con una lista que contiene la descripción y la cantidad de cada producto
- **03-lista-con-imagen-descripción-y-cantidad:** Tercera entrega con una lista que contiene la imagen, descripción y cantidad de cada producto.

De hecho, este tipo de abordaje funciona, sin embargo, tenga en cuenta que para cada una de las versiones que surjan tendremos que realizar este proceso manual varias veces. Si observamos bien, este abordaje tiende a ser muy laborioso y está repleto de riesgos, tales como:

- **Control:** para cada cambio de proyecto, tendremos que recordar crear una carpeta de proyecto actual.
- **Historial:** Es muy difícil tener una visión amplia y objetiva de lo que se hizo y cuándo se hizo, así como de lo que cambió de una versión a otra.
- **Reversión:** si necesitamos **volver solo una parte** de lo que se ha hecho hace mucho tiempo, manteniendo las otras cosas nuevas que se han implementado, probablemente vamos a tener un enorme trabajo.

¿Existe una forma más sencilla y objetiva de afrontar este tipo de situación?

## Sistemas de control de versiones

Para hacer frente a estas situaciones de forma más eficaz, podemos utilizar sistemas de **control de versión**, también conocido como **VCS** del inglés *Version Control System*, o traducido como, Sistema de Control de Versión.

Un VCS ampliamente utilizado es el **Git**. Una herramienta muy común en la comunidad para versionar proyectos que mantienen archivos fuentes como el nuestro:

*"Genial, pero [¿cómo puedo usar Git?](#)"*

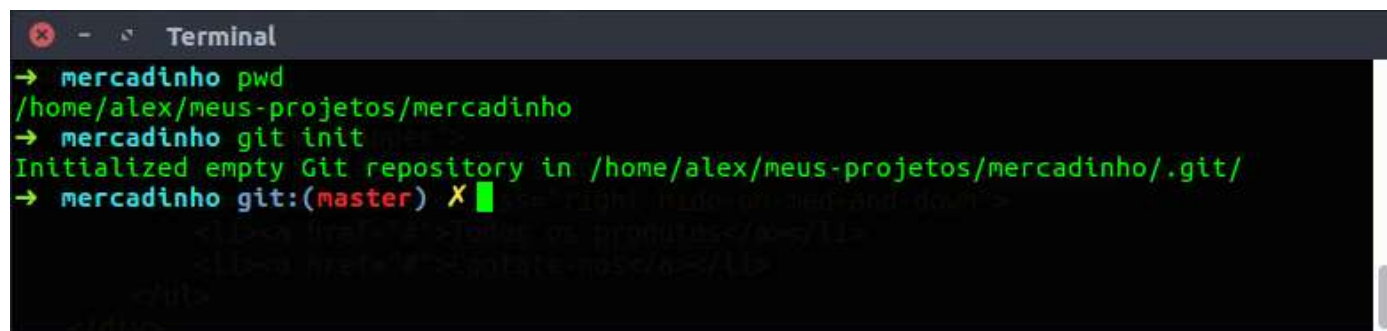
El primer paso de todo es instalarlo en tu computadora, puedes hacer esto en esta [página de descargas](#).

*Es posible instalar a través del administrador de paquetes, como Apt, Brew, Scoop, etc.*

## Iniciando Git

Después de la instalación, ¡ya podemos hacer el control de versión de nuestro proyecto! Para ello, por el terminal o prompt, nos dirigimos al directorio donde está el proyecto, en mi caso está **/home/alex/meus-projetos/mercadinho**.

Luego le decimos a Git que comience (init) la gestión del proyecto:`git init`

A screenshot of a terminal window titled "Terminal". It shows a series of commands and their outputs. The first command is `mercadinho pwd`, which outputs `/home/alex/meus-projetos/mercadinho`. The second command is `mercadinho git init`, which outputs `Initialized empty Git repository in /home/alex/meus-projetos/mercadinho/.git/`. The third command is `mercadinho git:(master) X`, where the 'X' is a red square icon. The terminal has a dark background with green and red text.

Observa que se exhibe un mensaje que indica que se ha iniciado un repositorio de Git vacío.

En otras palabras, ahora mismo creamos un **repositorio local** de Git dentro de nuestro proyecto, por lo que comenzó a gestionarlo.

*"Genial, pero ¿qué vamos a hacer ahora?"*

Primero, volveremos a nuestro proyecto a la primera versión que se entregó al cliente, luego, comenzaremos a crear las versiones que surgieron según la necesidad del cliente. ¡Entonces usaremos Git para que nos gestione eso!

## Agregando archivos para guardar

Volvemos a lo que teníamos desde el principio, así que primero debemos pedir a Git que **marque todos los archivos que queremos guardar** en esta primera versión.

Siguiendo los archivos que tenemos en el proyecto:



```
Terminal
→ mercadinho git:(master) X ls
css  js  produtos.html
→ mercadinho git:(master) X
```

Entonces podemos pedir a Git que agregue (add) nuestro archivo productos.html para guardarlo:

```
git add productos.html
```

¡Listo! Con eso, nuestro archivo fue marcado por Git, sin embargo, aún necesitamos marcar los archivos que están en el directorio css y js. Pero para hacer esto ¿necesito agregar todos los archivos contenidos en ellos uno por uno?

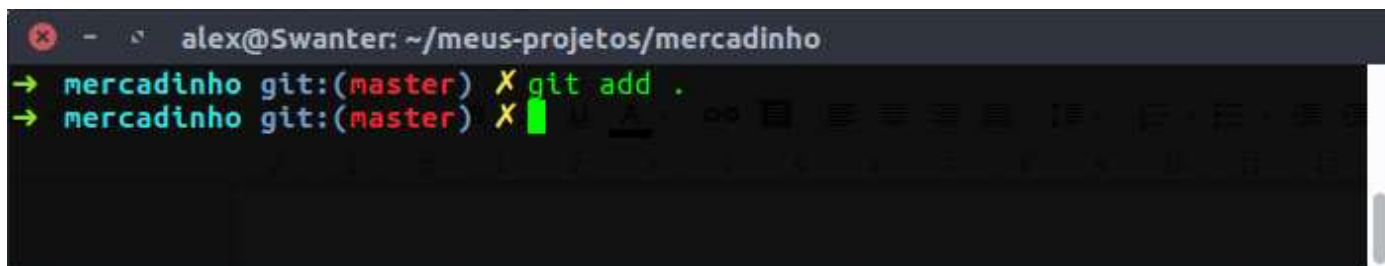
No necesariamente, puede agregar un archivo a la vez informando la ruta, pero si desea agregar todos los archivos de un directorio, simplemente díglele esto a Git:

```
git add css/
```

De esta forma, todos los archivos dentro de la carpeta css/ se han agregado a Git. Pero si en mi proyecto tengo muchos archivos y directorios, ¿necesito agregarlos cada uno a la vez?

## Agregar todos los archivos a la vez

Considerando que necesitamos **hacer eso en todos**, podemos decirle a Git que agregue (add) todos los archivos modificados (.):git add.



```
alex@Swanter: ~/meus-projetos/mercadinho
→ mercadinho git:(master) X git add .
→ mercadinho git:(master) X
```

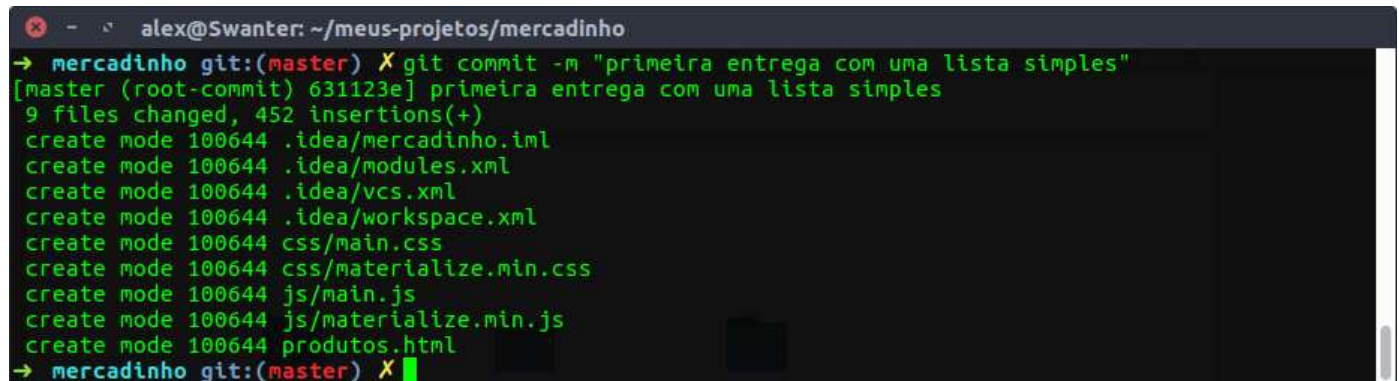
## Guardar versiones del proyecto



Ahora que hemos agregado todos los archivos que queremos guardar, debemos pedirle a Git que guarde el estado actual de nuestro proyecto, es decir, ¡nuestra primera versión!

En otras palabras, le indicaremos a Git que todo lo que le pedimos que guarde representa nuestra primera versión. Para esto usamos el comando `git commit`. Sin embargo, este comando requiere que enviemos un mensaje (-m) indicando lo que significa este estado.

```
git commit -m "primera entrega con una lista sencilla"
```

A terminal window with a dark background. The prompt is 'alex@Swanter: ~/meus-projetos/mercadinho'. The command 'git commit -m "primeira entrega com uma lista simples"' is entered. The output shows the commit hash '[master (root-commit) 631123e]', the commit message 'primeira entrega com uma lista simples', and a list of 9 files created with their modes (100644) and paths: .idea/mercadinho.iml, .idea/modules.xml, .idea/vcs.xml, .idea/workspace.xml, css/main.css, css/materialize.min.css, js/main.js, js/materialize.min.js, and produtos.html. The prompt returns to 'mercadinho git:(master) X'.

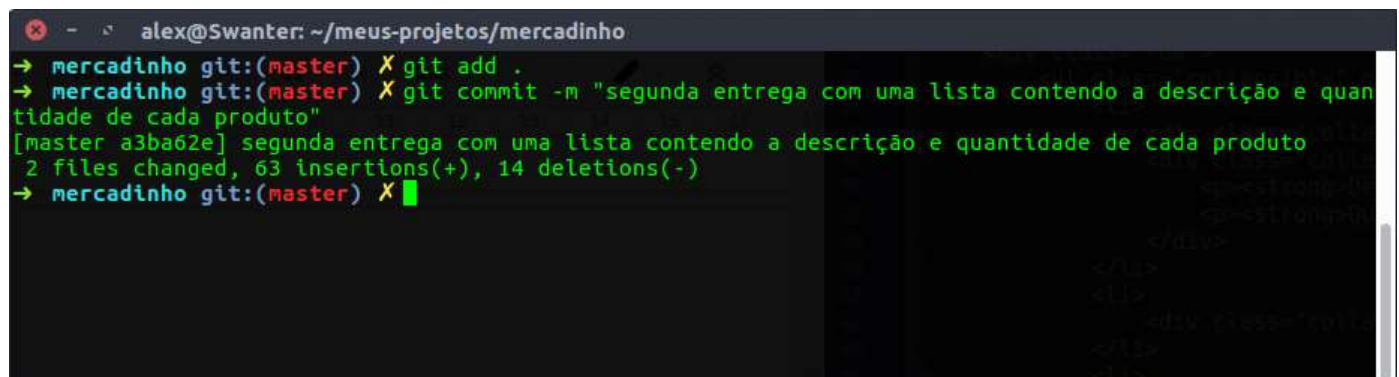
*Si no usamos este parámetro con un mensaje específico, Git solicita al sistema que abra un editor de texto estándar para que el mensaje se asocie de la misma manera.*

¡Note que Git mostró todos los archivos que se agregaron en esta versión que guardamos!

Ahora que tenemos nuestra primera versión, ajustemos el proyecto y coloquemos las otras versiones que solicitó el cliente y hagamos el mismo procedimiento, es decir, marquemos los archivos a guardar con el `add` y luego guardemos la versión con el `commit`.

## Ingresando la descripción y la cantidad

En la segunda versión modifiqué el HTML para que contara con las listas con descripción, por lo que tenemos el siguiente resultado al **comitar** (no se sorprenda, es común que hablemos así):

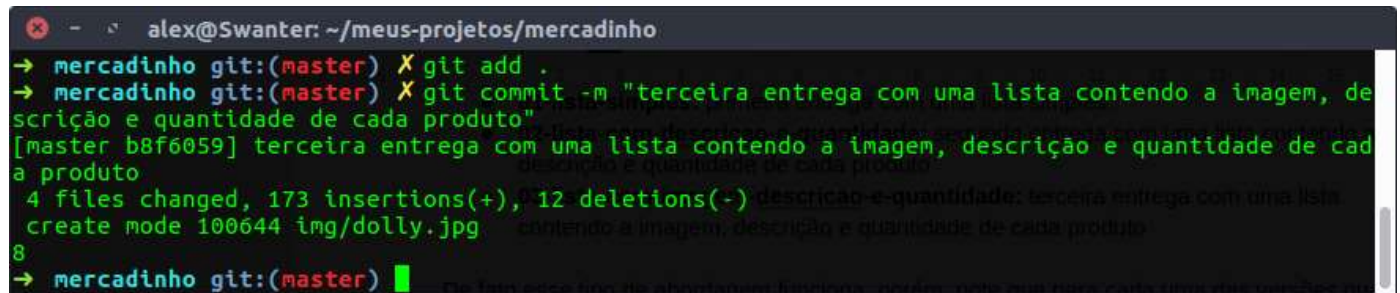
A terminal window with a dark background. The prompt is 'alex@Swanter: ~/meus-projetos/mercadinho'. The command 'git add .' is entered. Then, the command 'git commit -m "segunda entrega com uma lista contendo a descrição e quantidade de cada produto"' is entered. The output shows the commit hash '[master a3ba62e]', the commit message 'segunda entrega com uma lista contendo a descrição e quantidade de cada produto', and the statistics '2 files changed, 63 insertions(+), 14 deletions(-)'. The prompt returns to 'mercadinho git:(master) X'.

Ves que ahora solo notificó que **hubo un cambio en 2 archivos**, también esta vez no mostró que el archivo fue agregado, ¡solo registró lo que lo cambió! Es decir, 63 líneas insertadas y 14 eliminadas.

Además, observe que incluso con 9 archivos dentro del proyecto como vimos inicialmente, Git ya es lo suficientemente inteligente como para saber que cuando lo hicimos git add, necesitaba agregar solo los archivos que se modificaron :)

## Agregar una imagen al producto

Por fin, agregaré la imagen al producto y realizaré el mismo procedimiento:



```
alex@Swanter: ~/meus-projetos/mercadinho
→ mercadinho git:(master) X git add .
→ mercadinho git:(master) X git commit -m "terceira entrega com uma lista contendo a imagem, descrição e quantidade de cada produto"
[master b8f6059] terceira entrega com uma lista contendo a imagem, descrição e quantidade de cada produto
4 files changed, 173 insertions(+), 12 deletions(-)
create mode 100644 img/dolly.jpg
8
→ mercadinho git:(master)
```

Ves ahora, además de editar los archivos que ya se habían agregado una vez, también agregamos la imagen que no existía hasta ahora y Git indicó que se insertó específicamente en esa versión. En este punto, es posible que se esté preguntando:

*Genial, pero ¿cómo puedo ver si mis estados, o más bien mis commits, realmente fueron registrados?*

## Ver los commits

Para ver todos los commits que hemos realizado hasta ahora en el repositorio Git, simplemente use el comando `git log`:



```

git log
commit b8f60597f4a628fc9c64b69f9b7a005fb08d029d
Author: alexfelipe <alexfelipevieira@gmail.com>
Date:   Fri Jul 28 14:31:54 2017 -0300
    terceira entrega com uma lista contendo a imagem, descrição e quantidade de cada produto

commit a3ba62e6dbb089616b5047a1356a696d29079182
Author: alexfelipe <alexfelipevieira@gmail.com>
Date:   Fri Jul 28 12:56:27 2017 -0300
    segunda entrega com uma lista contendo a descrição e quantidade de cada produto

commit 631123e4d6276b49059a7ae96b1de86c928f71c2
Author: alexfelipe <alexfelipevieira@gmail.com>
Date:   Fri Jul 28 11:24:05 2017 -0300
    primeira entrega com uma lista simples
~
(END)

```

Ten en cuenta que Git muestra tanto el autor como la fecha de creación de la versión, por lo que tenemos la capacidad de ver el historial de cambios que se realizaron en el proyecto, como quién lo hizo y cuándo se realizó.

Además, observe que cada commit tiene un [hash](#) para identificarlo, en otras palabras, ¡es a través de estos hash que tenemos la capacidad de saber qué ha hecho un determinado commit o incluso navegar entre los commits!

## Navegando entre commits

Por ejemplo, para volver ahora a la primera versión usamos el comando `git checkout 631123e4d6276b49059a7ae96b1de86c928f71c2`, mira esto:

```

alex@Swanter: ~/meus-projetos/mercadinho
→ mercadinho git:(master) git checkout 631123e4d6276b49059a7ae96b1de86c928f71c2
Note: checking out '631123e4d6276b49059a7ae96b1de86c928f71c2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 631123e... primeira entrega com uma lista simples
→ mercadinho git:(631123e)

```

Usamos este hash porque se refiere al primer commit que guardamos, es decir, nuestra primera versión.

Si revisamos nuestro directorio:

```
alex@Swanter: ~/meus-projetos/mercadinho
→ mercadinho git:(master) git checkout 631123e4d6276b49059a7ae96b1de86c928f71c2
Note: checking out '631123e4d6276b49059a7ae96b1de86c928f71c2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 631123e... primeira entrega com uma lista simples
→ mercadinho git:(631123e) ls
css  js  produtos.html
→ mercadinho git:(631123e)
```

Mira, ya no tenemos la imagen que se agregó con el directorio **img**. En este punto, podrías estar pensando:

*"¿Pero cómo vuelvo al commit que tenía la imagen con las descripciones?"*

¡Sencillo! Solo ejecutas el comando `git checkout master`:

```
alex@Swanter: ~/meus-projetos/mercadinho
→ mercadinho git:(631123e) git checkout master
Previous HEAD position was 631123e... primeira entrega com uma lista simples
Switched to branch 'master'
→ mercadinho git:(master) ls
css  img  js  produtos.html
→ mercadinho git:(master)
```

¡Fíjate que ahora volvió la imagen! En otras palabras, volvemos al último estado que se guardó, es decir, el último commit.

Vale la pena señalar que hicimos uso de este **master**, que al principio no tiene mucho significado para nosotros. Sin embargo, es un tema que va un poco más allá de nuestro primer contacto con Git, tal vez en una próxima publicación :)

## Para saber más

Además de un VCS, Git también se conoce como un SCM del inglés, Source Code Management o gestión de código fuente. Además de él, también existen algunas alternativas con el mismo objetivo:

- [SVN](#)

- [Mercurial](#)
- [perseverar](#)

Básicamente, estas herramientas son capaces de aplicar todos los conceptos que vimos durante el texto, sin embargo, uno acaba teniendo alguna ventaja sobre el otro, así como la adopción de la comunidad de desarrolladores.

*Genial, pero ¿sobre cuál de estas herramientas vale la pena aprender?*

Una de las formas en que podemos verificar es exactamente qué tanto más se usa una herramienta en relación con otra. Para ello, tenemos la posibilidad de ver la popularidad de cada una de ellas [en esta encuesta realizada por RhodeCode en 2016](#).

Además de los temas que vimos anteriormente, un uso muy común en herramientas como Git es precisamente mantener nuestro proyecto disponible de forma remota para cualquier desarrollador que participe en el proyecto.

En otras palabras, dejar nuestro proyecto en un **repositorio en línea** como es el caso de [GitHub](#), [GitLab](#) o [BitBucket](#). Entonces, si necesita descargar el proyecto u obtener las últimas actualizaciones que se realizaron, ¡simplemente consulte estos repositorios remotamente!

El cambio es una característica común durante el desarrollo de un proyecto y, para manejar mejor este tipo de situación, tenemos los VCS como es el caso de Git.

Esta es una pequeña parte del mundo del control de versiones, también existen técnicas y conocimientos esenciales para utilizar estas herramientas de manera eficiente. Si desea aprender más sobre el control de versiones, aquí en Alura tenemos un [curso de Git](#).

Allá podrás ver tanto esta parte del control de versiones del proyecto como también técnicas más avanzadas que podemos aplicar durante el desarrollo en equipo, que es algo muy común en la vida cotidiana.

Puedes leer también:

- [Git y Github: que son y primeros pasos](#)
- [VisualStudio Code: instalación, teclas de acceso directo, plugins e integraciones](#)
- [SSH: acceso remoto a servidores](#)

ARTÍCULOS DE TECNOLOGÍA > DEVOPS

## En Alura encontrarás variados cursos sobre DevOps. ¡Comienza ahora!

**SEMESTRAL**

**US\$49,90**

un solo pago de US\$49,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 6 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**

[Paga en moneda local en los siguientes países](#)

**ANUAL**

**US\$79,90**

un solo pago de US\$79,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 12 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**

[Paga en moneda local en los siguientes países](#)

Acceso a todos  
los cursos

Estudia las 24 horas,  
dónde y cuándo quieras

Nuevos cursos  
cada semana

## NAVEGACIÓN

PLANES

INSTRUCTORES

BLOG

POLÍTICA DE PRIVACIDAD

TÉRMINOS DE USO

SOBRE NOSOTROS

PREGUNTAS FRECUENTES



## ¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

### BLOG

PROGRAMACIÓN

FRONT END

DATA SCIENCE

INNOVACIÓN Y GESTIÓN

DEVOPS

AOVS Sistemas de Informática S.A  
CNPJ 05.555.382/0001-33

### SÍGUENOS EN NUESTRAS REDES SOCIALES



### ALIADOS



En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

## CURSOS

### Cursos de Programación

Lógica de Programación | Java

### Cursos de Front End

HTML y CSS | JavaScript | React

### Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

### Cursos de DevOps

Docker | Linux

### Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics |  
Liderazgo y Gestión de Equipos | Startups y Emprendimiento